# LE NOUVEAU DEFI DE LA COORDINATION DES LANGAGES DE MODELISATION

*Gestion de l'hétérogénéité des modèles dans le développement et l'exécution de systèmes logiciels complexes*

Benoit Combemale[1], Julien De Antoni[2], Ali Koudri[3], et Jérôme Le Noir[3]
[1] Université de Rennes 1
[2] Université de Nice Sophia Antipolis
[3] Thales Research and Technology

***Résumé :*** *L'ingénierie dirigée par les modèles (IDM) vise à réduire la complexité accidentelle des développements de systèmes logiciels complexes. L'IDM vise à résoudre ce problème par l'utilisation de techniques de modélisation qui prennent en charge la séparation des préoccupations et la génération automatique des artefacts du système à partir de modèles (e.g., cas de tests, code d'implantation, scripts de déploiement et de configuration). Un modèle décrit un aspect du système et est généralement créé ou dérivé pour un objectif particulier.*
*Les modèles des différents aspects du système sont toutefois rarement manipulés indépendamment les uns des autres. Les ingénieurs systèmes sont donc confrontés à la tâche difficile de relier et synchroniser des informations issues de différents modèles. Les environnements modernes de conception et d'implantation de langages de modélisation offrent un bon support pour le développement d'un langage de modélisation, mais offrent peu ou pas de support pour permettre une utilisation coordonnée de plusieurs langages de modélisation. Supporter l'utilisation conjointe de plusieurs langages de modélisation pour la définition d'un système correspond au défi que nous qualifions de coordination des langages de modélisation, c'est-à-dire l'utilisation de plusieurs langages de modélisation pour soutenir le développement coordonné des aspects hétérogènes d'un système.*
*Dans cet article, nous faisons tout d'abord un bilan sur l'adoption de l'IDM et le constat de la multiplication des langages de modélisation dans les processus de développement industriels. Nous exposons ensuite le nouveau défi que constitue la coordination des langages de modélisation, et présentons les pistes de solutions actuellement explorées par les partenaires de l'initiative GEMOC.*

***Mots clés :*** *Ingénierie Dirigée par les Modèles, Langages de modélisation (dédié), Modélisation et simulation de modèles hétérogènes, Composition de modèle.*

## Un constat sur l'adoption des langages de modélisation

L'ingénierie dirigée par les modèles (IDM) vise à réduire la complexité accidentelle des développements de systèmes logiciels complexes [Schmidt06]. Une première source de complexité accidentelle vient du fossé important qui existe entre les concepts utilisés par les experts métiers et les abstractions de bas niveau fournies par les langages de programmation généralistes [France07]. Combler ce fossé manuellement est coûteux en termes de temps et d'efforts, en particulier lors de l'évolution des besoins et des technologies. L'IDM vise à résoudre ce problème par l'utilisation de techniques de modélisation qui prennent en charge la séparation des préoccupations et la génération automatique des artefacts du système à partir de modèles (*e.g.,* cas de tests, code d'implantation, scripts de déploiement et de configuration). Un modèle décrit un aspect du système et est généralement créé ou dérivé pour un objectif particulier. La séparation des préoccupations est soutenue par l'utilisation de différents langages de modélisation, chacun fournissant des concepts basés sur des abstractions qui sont spécifiques à un aspect du système. Par exemple, les réseaux de Petri stochastiques peuvent être utilisés pour créer des modèles de performance, alors que le formalisme fourni par l'outil Simulink peut être utilisé pour construire des modèles de simulation. Les technologies de l'IDM apportent

également un support à la manipulation de modèles, *e.g.,* requête, transformation, composition, analyse, exécution. Les langages de modélisation sont donc au cœur de l'IDM.

L'intégration de concepts dédiés et d'expériences de développement de haute qualité dans des langages de modélisation peut améliorer considérablement la productivité des développeurs et la qualité du système. Cette prise de conscience a conduit à travailler, à partir de la fin des années 90, sur des environnements supportant la conception et l'implantation de langages de modélisation dédiés (ou *Domain-Specific Modeling Languages* - DSML - en anglais) et des outils associés (*e.g.,* éditeurs, générateurs, outils d'analyse). Les environnements modernes d'IDM permettent aujourd'hui de concevoir et d'implanter des DSMLs qui sont utilisés pour créer des modèles qui jouent un rôle essentiel dans les différentes phases d'un développement. Des environnements tels que Microsoft DSL Tools[1], Eclipse Modeling Framework[2] et MetaEdit+[3] supportent la spécification de la syntaxe abstraite, de la syntaxe concrète, et des sémantiques statique et dynamique d'un DSML. Ces environnements visent ainsi à répondre aux besoins des développeurs de DSMLs dans un large spectre de domaines d'application. Un DSML offre un pont entre l'espace dans lequel les experts métiers travaillent et l'espace de mise en œuvre (implantation). Les domaines dans lesquels les DSMLs ont été développés et utilisés comprennent entre autres les systèmes embarqués critiques tels que l'automobile et l'avionique, et les systèmes cyber-physiques.

Une étude récente réalisée par Hutchinson et al. [Hutchinson11] fournit quelques indications sur le fait que les DSMLs jouent un rôle central dans l'adoption industrielle de l'IDM. La recherche sur le développement systématique de DSMLs a produit une base technologique qui est maintenant assez robuste pour soutenir l'intégration de processus de développement de DSML dans les environnements de développement de systèmes industriels complexes. C'est par exemple le cas au sein de la société Thales qui a défini une méthodologie adaptée à ses enjeux d'ingénierie (ARCADIA [ARCADIA08]), et un environnement de modélisation associé offrant le support à l'implémentation de vues d'ingénieries de spécialité, la transformation de modèle conservative, la gestion multi-utilisateur, la gestion de configuration, l'analyse d'impact, etc. Un tel environnement de modélisation est constitué de 20 langages et d'environ 400 concepts dédiés. Les ingénieurs de Thales utilisent cet environnement pour la conception et l'analyse de systèmes complexes nécessitant des compromis entre diverses préoccupations issus de domaines d'expertises différents. Chaque expert impliqué dans le développement de tels systèmes possède ses propres représentations du système avec pour chacune de ces représentations une sémantique métier particulière.

Dans le développement d'un système logiciel complexe, plusieurs DSMLs sont utilisés pour capturer les aspects hétérogènes du système. Par exemple, la conception d'un radar va faire intervenir des experts du traitement du signal, des experts du traitement des données, des ingénieurs logiciels et des ingénieurs matériels. Dans cet exemple, l'expert du traitement du signal pourra raisonner en termes de flots de données; l'expert du traitement des données

[1] Cf. http://www.microsoft.com/en-us/download/details.aspx?id=2379
[2] Cf. http://www.eclipse.org/modeling/emf
[3] Cf. http://www.metacase.com/mep

pourra raisonner en termes de flots de contrôle; les experts en développement logiciels pourront raisonner en terme de processus communicants alors que les experts en développement matériels pourront raisonner en termes de réseaux de calcul en pipelines.

Les modèles des différents aspects du système sont toutefois rarement manipulés indépendamment les uns des autres. Les ingénieurs systèmes sont donc confrontés à la tâche difficile de relier des informations issues de différents modèles. Par exemple, un ingénieur système peut avoir besoin d'analyser une propriété système qui requiert des informations dispersées dans des modèles exprimés avec différents DSMLs. Les environnements modernes de conception et d'implantation de DSMLs offrent un bon support pour le développement d'un DSML, mais offrent peu ou pas de support pour permettre une utilisation coordonnée de plusieurs DSMLs. Ce manque de support rend difficile le raisonnement sur des informations issues de modèles décrivant des aspects différents du système. Supporter l'utilisation conjointe de plusieurs DSMLs pour la définition d'un système correspond au défi que nous qualifions de *coordination des langages de modélisation*, c'est-à-dire l'utilisation de plusieurs langages de modélisation pour soutenir le développement coordonné des aspects hétérogènes d'un système.

## Besoin de coordination des langages de modélisation

Les travaux existants sur les langages de modélisation se sont concentrés sur la mise en place de langages permettant de combler le fossé entre un problème et son implantation. Une nouvelle génération de systèmes logiciels complexes (*e.g.,* systèmes de transport intelligents, *smart grid*, gestion de l'énergie des bâtiments, etc.) présentent de nouvelles opportunités pour tirer parti des langages de modélisation. Le développement de ces systèmes nécessite une expertise dans une variété de domaines de spécialité. Cette diversité implique une multiplication des acteurs qui doivent travailler de manière coordonnée sur les différents aspects du système et pendant les différentes phases du développement. Les DSMLs peuvent être utilisés pour soutenir le travail des experts métier travaillant sur un aspect du système, mais ils doivent aussi fournir les moyens de coordonner le travail entre les équipes travaillant sur différents aspects et à différentes phases du développement. Dans la vision où les DSMLs seraient coordonnés, leur intégration fournit les moyens de déterminer comment le travail sur un aspect particulier impacte les autres aspects. L'objectif est de fournir un appui pour véhiculer les informations pertinentes ainsi que pour coordonner les activités de développement et les technologies associées au sein et entre les équipes. L'objectif est également de fournir un soutien au contrôle des artefacts de développement produits par plusieurs équipes.

La séparation des préoccupations et des problèmes de coordination évoqués ci-dessus ont été au cœur du génie logiciel depuis les premiers travaux sur la modularisation du logiciel [Conway68] [Parnas72]. La modularité dans le développement de systèmes logiciels conduit à des problèmes de coordination bien connus [Cataldo09], comme par exemple les problèmes liés à la coordination des travaux sur la distance temporelle, géographique ou socio-culturelle [Herbsleb99]. Cela a conduit à l'émergence de travaux pour une globalisation du génie logiciel, dont la coordination socio-technique, y compris la coordination des acteurs et des technologies qu'ils utilisent pour mener leurs activités de développement, constitue un défi majeur

[Herbsleb07]. Dans ce contexte, les DSMLs peuvent être utilisées pour soutenir la coordination socio-technique en fournissant les moyens pour les acteurs du développement de combler le fossé entre la façon dont ils perçoivent un problème, et les technologies de programmation utilisées pour mettre en œuvre une solution. Les DSMLs peuvent également être utilisés pour appuyer la coordination du travail entre plusieurs équipes si elles sont soutenues par des mécanismes pour la spécification et la gestion de leurs interactions. Cette prise de conscience est nécessaire pour minimiser l'isolement social contre-productif qui peut se produire lorsque le travail est réparti entre différentes équipes, chacune utilisant leurs propres concepts [Souza04].

De nombreux types de relations peuvent exister entre des DSMLs pour décrire les diverses interactions possibles entre modèles hétérogènes. Nous illustrons certaines relations et leurs impacts grâce à une vue simplifiée d'un exemple de traitement de données radar développé chez Thalès. Une vue simplifiée de l'architecture de ce système est représentée sur la figure 1. Si l'on considère tous les éléments de cette architecture en boîte noire (comme le composant *Tracking*) alors on voit apparaitre un système flot de donnée composé de 4 composants dont le but est de balayer l'environnement électromagnétique afin d'y détecter des objets, de les identifier et de prendre les mesures appropriées le cas échéant. Pour ce faire, le composant *DwellManagement* décide de la forme d'onde à émettre en fonction de l'historique et il envoie une donnée qui contient la description des impulsions électromagnétiques à émettre au composant *AntennaHitDetection*. Ce dernier utilise les données reçues pour générer des signaux radio fréquence à destination de l'environnement électromagnétique (incluant une conversion numérique / analogique). Ce même composant reçoit ensuite les échos, les traduit en données numériques (conversion analogique / numérique), puis extrait les échos correspondants aux *bursts* qui ont été émis, les caractérise (point position, vitesse) et les stocke. Le composant *TrackDetection* utilise alors ces données pour détecter si les échos correspondent à un seul et même gros objet ou si il s'agit de deux ou plus petits objets. Ces données sont analysées par le composant *Tracking* qui détermine si les objets vus dans l'espace électromagnétique constituent une menace ou non en fonction de ce qui a été vu auparavant. Le composant *Tracking* met à jour la base de données des *Tracks* : mise à jour des positions des objets déjà identifiés, suppression des objets qui ont disparus de l'espace électromagnétique, etc. Enfin, le composant *DwellManagement* utilise ces informations comme étant son historique.

Les composants intervenant dans ce système sont de natures différentes. Les composants *AtennaHitDetection* et *Tracking* sont orientés flot de données alors que les autres composants sont orientés flot de contrôle. Ces composants sont donc décrits par des DSMLs différents. Les interactions entre les différents composants doivent donc être décrite sous la forme de relations entre les différents DSMLs utilisées, explicitant ainsi leur coordination dans le processus de développement de tels systèmes.

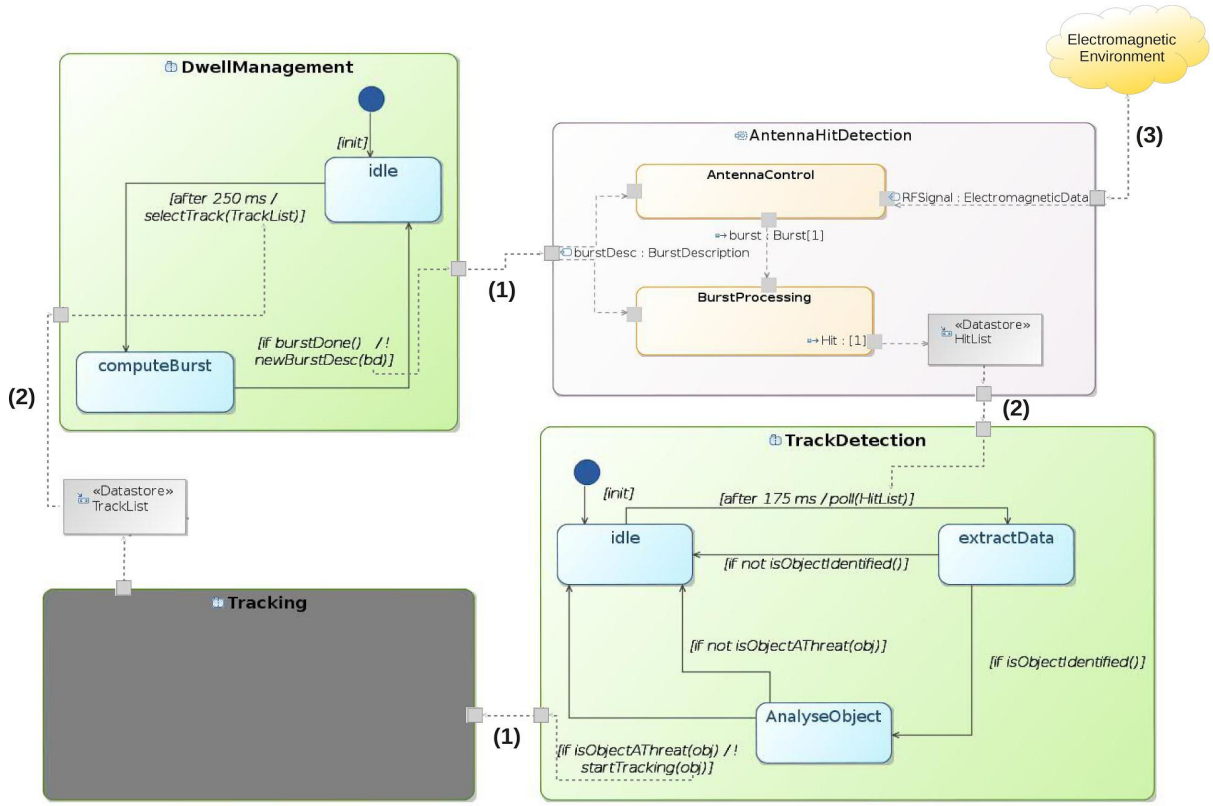


Figure 1: Exemple simplifié d'un système radar

Cet exemple permet d'identifier deux types de relations entre DSMLs : la composition et le raffinement. Toutefois il est nécessaire de préciser la sémantique de ces relations qui peuvent être de natures diverses. Par exemple une composition peut être structurelle ou comportementale. Dans le cas d'une composition comportementale, plusieurs sémantiques peuvent également exister en fonction de la nature des DSMLs impliqués. Par exemple, sur la figure 1, les liens notés (1) représentent une composition où la réception d'une donnée déclenche un comportement et où une synchronisation orientée apparaît entre les modèles. Le lien noté (3) étant bi-directionnel, on obtient une synchronisation plus fine des différents modèles; alors que les liens notés (2) représentent des échanges de données n'impliquant pas de synchronisation entre les modèles. Les relations de raffinement peuvent elles aussi être de différents types. Dans l'exemple fourni, le comportement d'un composant est décrit par un autre DSML. Un autre type de raffinement consiste à élargir l'expressivité d'un langage pour décrire un même modèle et par le fait rajouter des informations (par exemple passer d'un langage à base de *token* à un langage décrivant les données échangées, leurs tailles, etc.).

Enfin, sur l'exemple illustré sur la figure 1, certains des éléments sont implémentés dans le domaine de l'ingénierie logicielle et d'autres dans le domaine de l'ingénierie matérielle. Ce type de composition représente une **allocation** et représente une composition possiblement complexe changeant profondément le parallélisme potentiel et les différentes synchronisations au sein du système. Les allocations peuvent également restreindre les communications entre les composants et intervient comme une restriction des comportements décrit par les DSMLs.

Au travers de cet exemple simple, il apparaît indispensable de pouvoir formaliser explicitement les relations possibles entre DSMLs et régissant les interactions entre modèles hétérogènes. Après le succès de l'adoption des DSMLs dans les processus de développement de systèmes complexes, leur multiplication soulève le nouveau défi de leur coordination. Ce défi est au cœur des travaux réalisés au sein de la nouvelle initiative GEMOC (http://gemoc.org), et fait l'objet d'un projet ANR du même nom explorant la coordination comportementale au travers des pistes de solutions présentés dans la section suivante.

# Vers une convergence de la théorie des langages et de la théorie de la concurrence pour la coordination comportementale des DSMLs

La figure 1 a permis d'illustrer dans la section précédente différentes interactions possibles entre les différents sous systèmes, et donc potentiellement des relations différentes entre les divers langages de modélisation. Lorsque ces différentes relations ont été mise en œuvre au sein d'outils existants (*e.g.*, Raphsody), il est apparu que de telles relations de composition furent encodées dans le langage associé à l'outil, introduisant ainsi une complexité accidentelle dans le modèle composé. L'analyse a posteriori nous fait dire que cette complexité est du au manque d'opérateur de composition permettant d'une part de capitaliser les compositions classiques d'un domaine (éventuellement en assurant la conservation de certaines propriétés) et d'autre part de décrire la composition sans polluer le modèle initial.

Afin de permettre la coordination de DSMLs hétérogènes au travers de relations explicites, il est nécessaire que chacun des DSMLs expose les informations nécessaires à l'expression de la relation de composition des modèles qui en sont conformes. Les informations à exposer doivent être choisies avec parcimonie pour éviter de complexifier la définition de la composition en se préoccupant d'informations non essentielles. Cependant, les informations exposées doivent être suffisantes et ne pas restreindre les possibilités de composition souhaitées. En particulier il est important de se poser les questions suivantes : "Comment modulariser la définition d'un langage pour faciliter sa coordination avec d'autres ?"; "Comment spécifier les points d'interactions qu'un langage expose ?"; "Comment des points d'interaction spécifiques peuvent être utilisés pour mettre différents langages en relation ?". Nous explorons dans la suite de cette section comment les théories des langages et de la concurrence pourraient permettre d'apporter des éléments de réponses aux questions précédentes.

Un des principaux challenges associé à la coordination des DSMLs concerne l'intégration de leurs sémantiques. Une telle intégration est nécessaire pour permettre, par exemple, l'analyse de propriétés globales du comportement du système ou encore pour pouvoir effectuer sa simulation. Afin de permettre cette intégration, il est nécessaire d'offrir un mécanisme unifié pour exprimer des points d'interaction sémantique dédiés au niveau des langages, et sur lesquels pourra s'appuyer un mécanisme unifié pour l'expression de relations de composition spécifiques entre ces langages.

Afin de permettre la définition d'opérateur de composition comportementaux entre langages, il est nécessaire de comprendre comment la sémantique comportementale de tels langages est définie. Dans les 50 dernières années, de nombreuses approches issues de la théorie des langages ont été proposées pour définir (formellement) la sémantique des éléments de langages (*e.g.*, sémantique opérationnelle, translationnelle, axiomatique). Ces approches spécifient comment un domaine syntaxique est dénoté dans un domaine sémantique. Chacune des approches a ses avantages et ses défauts mais aucune d'entre elles ne fournit un cadre de raisonnement commun et global des aspects liés à la concurrence [Winskel93]. Pourtant, l'évolution des ressources de calcul actuelles, qui deviennent de plus en plus parallèles et massivement distribuées, introduit des contraintes qu'il est nécessaire de prendre en compte lors de la composition de comportements. Gérer une telle évolution nécessite d'arrêter de penser en premier lieu de manière séquentielle comme le font la plupart des langages de programmation actuels. À la place, les DSMLs doivent bien sûr tirer partie des résultats issus de la théorie des langages mais aussi des résultats issus de la théorie de la concurrence. La théorie de la concurrence se focalise sur l'étude du parallélisme potentiel d'un programme et fût initiée par Hoare avec CCS (Calculus of Communicating System [Milner82]), Milner avec CSP (Communicating Sequential Processes [Hoare78]) et Petri avec les réseaux du même nom. Ces approches fournissent des langages possédant des opérateurs explicites permettant de décrire le parallélisme et les synchronisations. De telles approches représentent également les premiers essais visant à capturer de manière explicite le modèle de calcul (MoC). Ces approches ont été largement étudiées et raffinées afin de spécifier le parallélisme du calcul. De là ont émergés différents cadres de raisonnement qui abstrait les interactions d'un système (event structure [Winskel82], tag signals [Lee98], trace theory [Mazurkiewicz87]). Tous ses modèles ont mis l'accent sur les relations de causalités et de temporalités entre les événements significatifs d'un système. Expliciter de telles relations a permis de raisonner sur l'ordre partiel des interactions dans un système (potentiellement massivement parallèle). Cependant, le côté traitement séquentiel des données n'est pas détaillé par de telles approches.

Que ce soit le domaine de la théorie des langages ou le domaine de la théorie de la concurrence, chacun, séparément, a fourni des contributions significatives pour la conception, l'implantation et la coordination de langages. Cependant, il manque encore une fertilisation croisée de ces deux domaines. L'IDM apparaît comme un terrain naturel permettant de favoriser cette fertilisation croisée et ainsi de tirer partie du meilleur de chacun des domaines. Cette fertilisation croisée peut prendre différentes formes en fonction de l'intention de modélisation ; cependant un langage de modélisation devrait devenir un terrain de jeu naturel pour raisonner sur la concurrence, les structures de données, le typage et les interactions avec d'autres langages. Un tel environnement devrait définir quels sont les constituants d'un tel langage de modélisation, et définir également comment la sémantique doit être structurée pour permettre d'exhiber ses points d'interactions.

En réfléchissant de cette manière, il semble que la fertilisation croisée de la théorie des langages et de la théorie de la concurrence doivent permettre l'articulation entre les règles d'évolution d'un programme et les aspects causaux et temporels qui définissent quand de telles évolutions peuvent/doivent être appliquées. Une articulation de ce type pourrait également être

réifiée comme un concept de première classe permettant de séparer les différents constituants d'un langage tout en assurant leur cohérence. Cette séparation explicite des préoccupations en œuvre dans un langage devrait contribuer à adresser le défi naissant qu'est la coordination des langages de modélisation.

# References


[Schmidt06] Douglas C. Schmidt: Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer 39(2): 25-31 (2006)

[France07] Robert B. France, Bernhard Rumpe: Model-driven Development of Complex Software: A Research Roadmap. FOSE 2007: 37-54

[Hutchinson11] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical assessment of MDE in industry. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 471-480.

[ARCADIA08] J.-L. Voirin, Method & Tools for constrained System Architecting, at INCOSE'08 Symposium

[Conway68] M.E. Conway, "How Do Committees Invent?" Datamation, Vol. 14, No. 4, Apr. 1968, pp. 28–31.

[Parnas72] D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," Comm. ACM, Vol. 15, No. 12, 1972, pp. 1053–1058.

[Cataldo09] Cataldo, M., Mockus, A., Roberts, J.A., & Herbsleb, J.D. (2009). Software Dependencies, Work Dependencies, and Their Impact on Failures. *IEEE Transactions on Software Engineering*, 99, 864-878.

[Herbsleb99] Herbsleb, J.D. and Grinter, R.E., Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software, Sept./Oct., (1999), p. 63-70.

[Herbsleb07] James D. Herbsleb. 2007. Global Software Engineering: The Future of Socio-technical Coordination. In *2007 Future of Software Engineering* (FOSE '07). IEEE Computer Society, Washington, DC, USA, 188-198.

[Souza04] C.R.B. de Souza, D. Redmiles, L. Cheng, D. Millen, and J. Patterson, "How a Good Software Practice Thwarts Collaboration —The Multiple Roles of APIs in Software Development," Proc. Conf. Foundations of Software Eng., pp. 221-230, 2004.

[Winskel93] Winskel, Glynn. The formal semantics of programming languages: an introduction. The MIT Press, 1993.

[Milner82] Milner, Robin. *A calculus of communicating systems*. Springer-Verlag New York, Inc., 1982.

[Hoare78] Hoare, Charles Antony Richard. "Communicating sequential processes." Communications of the ACM 21, no. 8 (1978): 666-677.

[Winskel82] Winskel, Glynn. Event structure semantics for CCS and related languages. Springer Berlin Heidelberg, 1982.

[Lee98] Lee, E.A., Sangiovanni-Vincentelli, A.: A Framework for Comparing Models of Computation. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 17(12), 1217–1229 (1998)

[Mazurkiewicz87] Mazurkiewicz, Antoni. "Trace theory." In Petri Nets: Applications and Relationships to Other Models of Concurrency, pp. 278-324. Springer Berlin Heidelberg, 1987